Using Object Files 26

Apple's Macintosh Programmer's Workshop is another common development environment for the Macintosh. It creates object files, or `.o` files, that you can add to your projects. The THINK Project Manager includes a translator, called the .o Converter, that converts MPW object files into a form that the THINK Project Manager can use.

Usually, you just need to add an object file to your project and let the .o Converter translate it for you. In some cases, however, you need to do some additional work. The first section of this chapter explains the additional work. The other sections of this chapter explain the limitations of the .o Converter and how to set options for the .o Converter.

What you should know

You need to know how to build project documents, use the THINK Editor, and compile a file in your project. To use a `.o` file that's over 32K in your project, you also need to have Apple's Macintosh Programmer's Workshop and know how to use it.

Contents

Adding MPW .o Files to Your Project 519
- Create the vocabulary file 519
- Edit the vocabulary file 521
- Create the final object code 521

Converting Large .o Files 521
- Create a list of symbols. 522
- Split up the symbols. 522
- Make the pieces 523
- Use the .o Converter to convert the pieces 523

Setting .o Converter Options 523
- The use Toolbox trap list option 524
- The use '.v' file option 524

Conversion Limits 525
Segmentation limits 525
Computed references 525
Reference records 525
SADE records 525
MPW run-time routines 525

Adding MPW .o Files to Your Project

Your project can contain an object file created by one of Apple's MPW compilers or assemblers. The .o Converter, in your `Translators` folder, converts the object file into object code that the THINK Project Manager can use and adds the converted code to your project file.

Usually, you can just add the `.o` file to your project and let the .o Converter translate it for you. However, if the `.o` file contains all uppercase function names, you need to do some additional work. The `.o` file will contain all uppercase names if it was created with MPW Pascal or with the MPW Assembler with the mixed case option turned off. These are the steps:

1. Add the `.o` file to your project.
2. Use the .o Converter to create a vocabulary file from the `.o` file.
3. Edit the vocabulary file to correct the function names.
4. Add to your project the object code created from the `.o` file and the edited vocabulary file.

Note

If you need to convert a `.o` file that is over 32K, first follow the directions in the section "Converting Large .o Files" later in this chapter to split the file. Then follow the directions here to convert the smaller files.

Create the vocabulary file

To create a vocabulary file from a `.o` file, follow these steps:

1. Add the `.o` file to your project.
2. Select the .o Converter options command from the **Edit** menu.
3. Check both the use Toolbox trap list and the use '.v' file check boxes, as in Figure 26-1. Click OK.

4. In your Project window, select your `.o` file, and choose **Compile** from the **Source** menu.

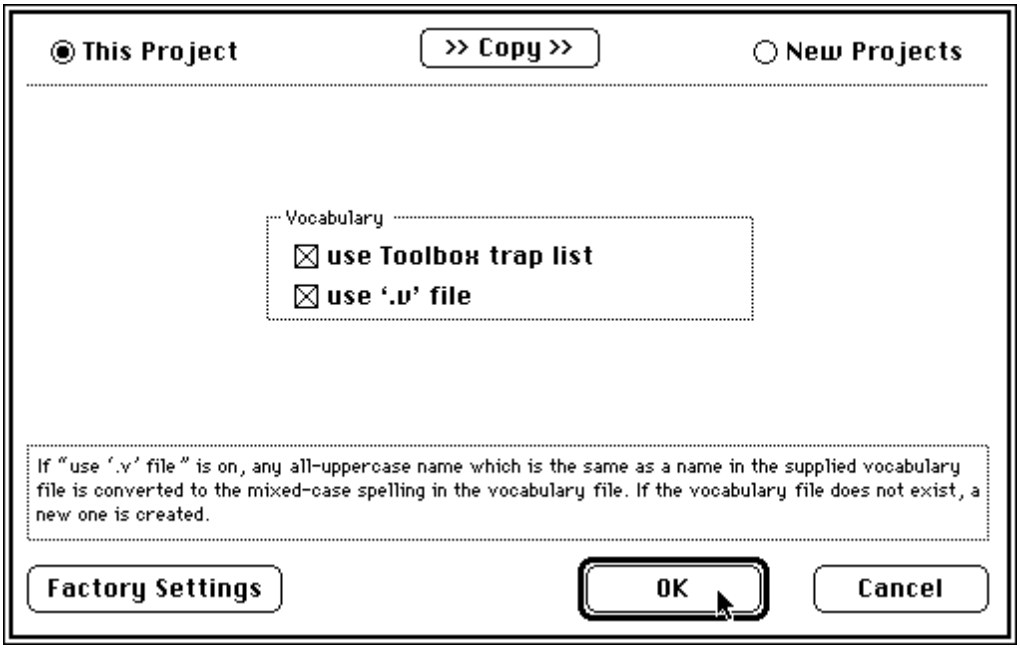

Figure 26-1 Setting the .o Converter options for an MPW .o file

The .o Converter notices that there is no vocabulary file so it creates one in the same folder as your `.o` file. It also adds object code to your project. Ignore the object code for now.

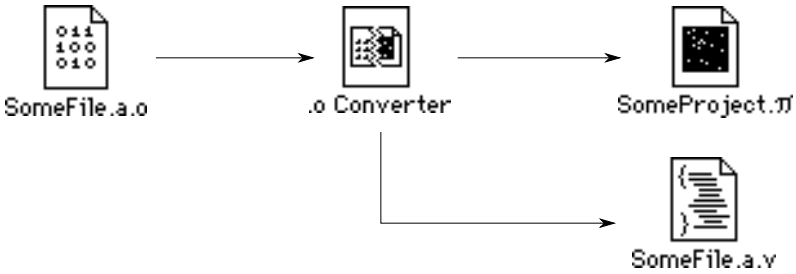

Figure 26-2 Creating a vocabulary file from an MPW .o file

Note

The .o Converter cannot convert some `.o` file records. If you get an error during this pass, see "Conversion Limits" later in this chapter to see if there is a way to work around the error.

Edit the vocabulary file

The vocabulary file contains the name of every external function in the `.o` file. In most cases, these names are in all uppercase. Use the THINK Editor or any other text editor to edit the vocabulary file so the function names have the correct uppercase and lowercase spellings.

Create the final object code

Now that you have a vocabulary file, the second part is a little simpler. In your Project window, select your `.o` file, and choose **Compile** from the **Source** menu.

In this second pass, the .o Converter uses the vocabulary file that you created and edited in the first pass. The resulting object code contains all the functions in the appropriate case.

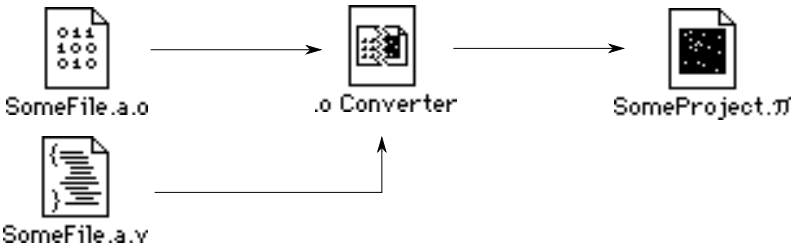

Figure 26-3 Creating object code from an MPW .o file

Converting Large .o Files

If you want to convert an object file that has more than 32K of code and data, you need to break it up into smaller object files that the .o Converter can handle. The best way to split up large object files is to use Apple's Macintosh Programmer's Workshop. The instructions in this section give you one way of splitting up large `.o` files.

Create a list of symbols

The first thing you need to do is find out the names of the symbols in the object file. Use the MPW `dumpobj` tool to do this. Suppose that the object file you want to convert is called `OFILE.o`. Use this MPW command to get a list of symbols:

```
dumpobj OFILE.o -d -h -i -n > OLIST.txt
```

The list in the output file, `OLIST.txt`, has size information about each symbol that you don't need to do the conversion. Make sure that `OLIST.txt` is in an MPW window, and then use this MPW command to clean up the file:

This is how to type the special characters:
∞ Option 5
• Option 8
≈ Option x
® Option r

```
replace -c ∞ /•≈[0-9]+ (≈)®1∞/ ®1 OLIST.txt
```

Split up the symbols

Now you need to split up the symbols into groups for the smaller object file. For example, suppose that `OLIST.txt` contains these symbols:

```
DrawThings
CalcThings
UtilThings
MemThings
```

To split up the large object file into three pieces, you need to create three lists. Each list should contain the names of the symbols you *don't* want in the smaller piece. Make these lists by copying `OLIST.txt` and editing it. Call the resulting lists `filter1.txt`, `filter2.txt`, and `filter3.txt`.

To break up the large object file into the three following pieces:

piece1.o	**piece2.o**	**piece3.o**
DrawThings	UtilThings	MemThings
CalcThings		

your filter files would look like this:

filter1.txt	**filter2.txt**	**filter3.txt**
UtilThings	DrawThings	DrawThings
MemThings	CalcThings	CalcThings
	MemThings	UtilThings

Make the pieces

Now you can use the filter lists to create the pieces from the original object file. You need to use the MPW `lib` tool for each piece:

```
lib OFILE.o -o piece1.o -df filter1.txt -Sym Off
lib OFILE.o -o piece2.o -df filter2.txt -Sym Off
lib OFILE.o -o piece3.o -df filter3.txt -Sym Off
```

Use the .o Converter to convert the pieces

Finally, follow the directions in section “Adding MPW .o Files to Your Project” earlier in this chapter to convert the smaller pieces.

Setting .o Converter Options

If you’re converting a `.o` file that isn’t from one of Apple’s MPW compilers or assemblers, you may need to set up the .o Converter options differently. This section explains in detail what the options do.

In some `.o` files, symbol names appear in all uppercase. The vocabulary mechanism provides a way to translate such names back to their proper capitalizations. Only names entirely in uppercase will

be translated. The .o Converter dialog, in Figure 26-4, lets you choose whether to use the vocabulary mechanism.

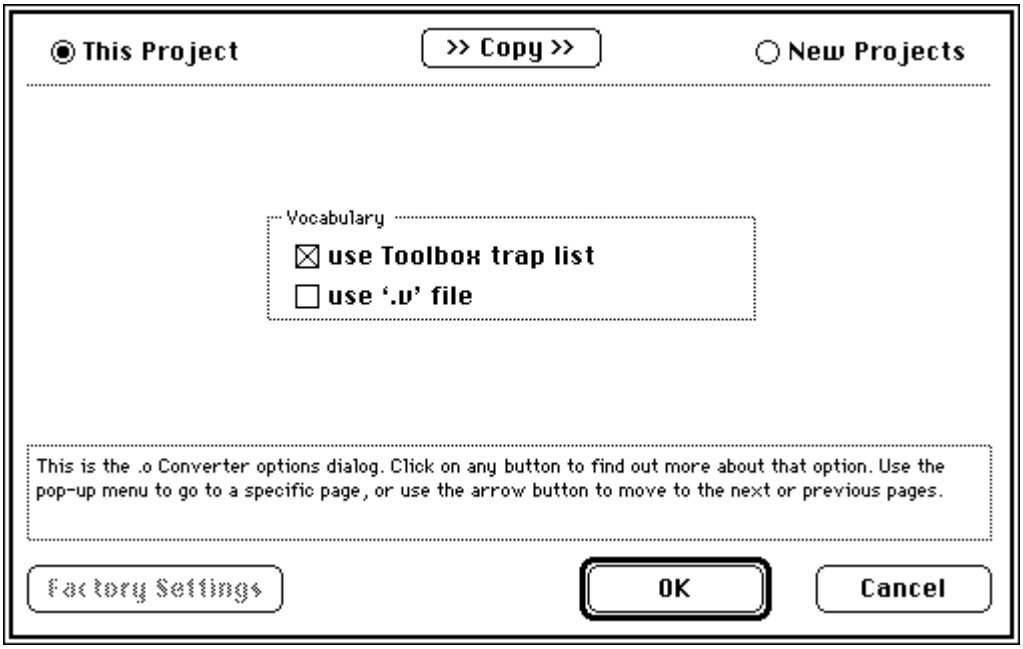

Figure 26-4 The .o Converter options dialog

The use Toolbox trap list option

When the use Toolbox trap list option is checked, the Toolbox glue routine names are added to the vocabulary. This includes all the routines in *Inside Macintosh*.

The use '.v' file option

When the use '.v' file option is checked, the translator looks for a vocabulary file for each `.o` file. The vocabulary file is a text file that contains the proper capitalization for each symbol in the `.o` file, one symbol per line. If the file to be converted is named `xyzzy.o`, the vocabulary file must be named `xyzzy.v`.

If the vocabulary file doesn't exist, the converter creates it. The file will contain one line for each symbol that appears entirely in uppercase in the `.o` file. You can edit this file to supply the proper capitalization, and then translate the file again.

Conversion Limits

Though the .o Converter can convert many object files, there are some object file records it can't convert.

Segmentation limits

The .o Converter ignores all segmentation directives in an object file, and places all the converted code into a single segment. The code and data in an object file are each limited to a total of 32K bytes.

Computed references

The .o Converter only converts computed references (object records of type 10) that are 16-bit modified references to code. It will not convert 8-bit or 32-bit computed references to code or 8-, 16-, or 32-bit references to data.

Reference records

The .o Converter can convert most reference records from high-level languages and many from assembly language. However, it cannot convert these types of references:

- 32-bit relative references generated by the `-m` switch in the MPW C compiler
- 32-bit relative references from assembly language
- 16-bit data-to-code or data-to-data references in assembly language

SADE records

The .o Converter ignores all SADE object file records.

MPW run-time routines

Sometimes an object file converts with no problems, but when you try to use it, you get an undefined symbols error message. If the object file was produced from a high-level language, like MPW C or MPW Pascal, these references are usually calls to run-time routines for long multiplication and division. To use object files that make calls to MPW run-time routines, you may need to extract the run-time routines from the MPW run-time libraries.